Deep Architectures for Joint Clustering and Visualization with Self-Organizing Maps

Florent Forest[1,2][0000−0001−6878−8752], Mustapha Lebbah[1][0000−0001−7245−6371], Hanane Azzag[1][0000−0001−6876−0688], and Jérôme Lacaille[2]

[1] Université Paris 13, Laboratoire d'Informatique de Paris-Nord (LIPN), 93430 Villetaneuse, France
[2] Safran Aircraft Engines, 77550 Moissy-Cramayel, France
forest@lipn.univ-paris13.fr

Abstract. Recent research has demonstrated how deep neural networks are able to learn representations to improve data clustering. By considering representation learning and clustering as a joint task, models learn clustering-friendly spaces and achieve superior performance, compared with standard two-stage approaches where dimensionality reduction and clustering are performed separately. We extend this idea to topology-preserving clustering models, known as self-organizing maps (SOM). First, we present the Deep Embedded Self-Organizing Map (DESOM), a model composed of a fully-connected autoencoder and a custom SOM layer, where the SOM code vectors are learnt jointly with the autoencoder weights. Then, we show that this generic architecture can be extended to image and sequence data by using convolutional and recurrent architectures, and present variants of these models. First results demonstrate advantages of the DESOM architecture in terms of clustering performance, visualization and training time.

Keywords: clustering · self-organizing map · representation learning · deep learning · autoencoder.

1 Introduction and related work

1.1 Joint Representation Learning and Clustering

Representations learned by deep neural networks are successful in a wide range of supervised learning tasks such as classification. Recent research has demonstrated how deep neural networks are able to learn representations to improve unsupervised tasks, such as clustering, in particular for high-dimensional data where traditional clustering algorithms tend to be ineffective. Most clustering algorithms, the most well-know example being the k-means algorithm, rely on similarity metrics (e.g. euclidean distance) that become meaningless in very high-dimensional spaces. The standard solution is to first reduce the dimensionality and then cluster the data in a low-dimensional space. This can be achieved by using, for example, linear dimensionality reduction techniques as Principal Component Analysis (PCA), or models with more expressive power such as

deep autoencoder neural networks (AE). In other words, this standard approach first optimizes a pure information loss criterion between data points and their low-dimensional embeddings (generally via a reconstruction loss between a data point and its reconstruction), and then optimize a pure clustering criterion (e.g. k-means quantization error). In contrast, recent *deep clustering* approaches treat representation learning and clustering as a joint task and focus on learning representations that are *clustering-friendly*, i.e. that preserve the prior knowledge of cluster structure.

One of the early approaches, Deep Embedded Clustering (DEC) [17], jointly learns representations and soft cluster assignments by optimizing a KL-divergence that minimizes within-cluster distance; IDEC [5] improves on this approach by optimizing the reconstruction loss jointly with the KL-divergence. The Deep Clustering Network (DCN) [18] combines representation learning with k-means clustering using an alternating training procedure to alternately update the autoencoder weights, cluster assignments and centroid vectors. A review of deep clustering is available in [1]. More recently, [3] overcame the non-differentiability of hard cluster assignments by introducing a smoothed version of the k-means loss.

Most recent approaches perform clustering using generative models such as variational autoencoders (VAE) with a gaussian mixture model (GMM) prior [8] or Wasserstein generative adversarial networks (WGAN) with GMM prior [6] and achieve state-of-the-art performance.

While the previously mentioned work do not make specific assumptions on the type of data and its regularities, other methods focus on specific types of data. For image data, it is common to use architectures based on convolutional neural networks (CNN) that leverage the two-dimensional regularity of images to share weights across spatial locations, as in [19] or [1]. Convolutional architectures can also be used for data with a one-dimensional regularity, such as (multivariate) time series. In this case, one-dimensional *causal* convolutions (also called *temporal* convolutions) are adapted. In particular, *dilated* convolutions are particularly successful in learning long-term dependencies, and even outperform recurrent LSTM networks on various tasks [2]. We are not aware of any application of these architectures for clustering, and will expose this idea in the last section. On the other hand, deep clustering of time series using an LSTM-based architecture was tackled in a recent unpublished work, Deep Temporal Clustering [14].

1.2 Joint Representation Learning and Self-Organization

We focus on a specific family of clustering algorithms called self-organizing maps, which perform simultaneous clustering and visualization by projecting high-dimensional data onto a low-dimensional map (typically two-dimensional for visualization purpose) having a grid topology. The grid is composed of *units*, also called *neurons* or *cells*. Each map unit is associated with a *prototype vector* from the original data space (also called *code vector*). Self-organizing map algorithms enforce a topological constraint on the map, so that neighboring units

on the map correspond to prototype vectors that are close in the original high-dimensional space. The most well-known self-organizing map model is Kohonen's self-organizing map (SOM) [10,11].

In this work, we propose several architectures for joint representation learning and self-organization with SOM. The main goals are to:

1. Learn the feature space and the SOM code vectors simultaneously, without using a two-stage approach.
2. Find a *SOM-friendly* space (using the term coined by [18]), i.e. a latent space that is more adapted to the SOM algorithm, according to some quality metric.

The SOM prototypes are learned in the latent space. To learn this new representation, we use an autoencoder neural network, composed of an encoder network that maps data points to the latent space, and a decoder network that reconstructs latent points into vectors of the original data space. For visualization and interpretation of the map, we need the prototypes to lie in the original feature space, so we reconstruct them using the decoder part of the autoencoder network. This approach very much resembles joint representation learning and clustering, but with an additional topology constraint. Our experiments show that using autoencoders with sufficiently high capacity yields meaningful low-dimensional representations of high-dimensional data that facilitate SOM learning and improve clustering performance, and that self-organization and representation learning can be achieved in a single joint task, thus cutting down overall training time.

To the best of our knowledge, the only other work performing joint representation learning with a SOM is the SOM-VAE model introduced in a recent unpublished work [4]. Their model is based on the VQ-VAE (Vector Quantization Variational Autoencoder) model which enables to train variational autoencoders (VAEs) with a discrete latent space [15]. [4] have added a topology constraint on the discrete latent space by modifying the loss function of VQ-VAE. However, there are many important differences between our DESOM model and SOM-VAE. First, SOM-VAE utilizes a discrete latent space to represent the SOM prototypes, whereas in DESOM, the SOM is learned in a continuous latent space. Secondly, they use a fixed window neighborhood to update the map prototypes, whereas we use a gaussian neighborhood with exponential radius decay. Finally, the DESOM model presented in this work is based on a deterministic autoencoder and not a VAE. Using a VAE in DESOM is left as future work.

We will first present our model with a generic, fully-connected, feed-forward autoencoder. The last sections will extend it to convolutional and recurrent architectures. Code is available at `https://github.com/FlorentF9/DESOM`.

2 DESOM: Deep Embedded SOM

We propose an approach where self-organization of the SOM prototypes and representation learning through a deterministic autoencoder are performed jointly

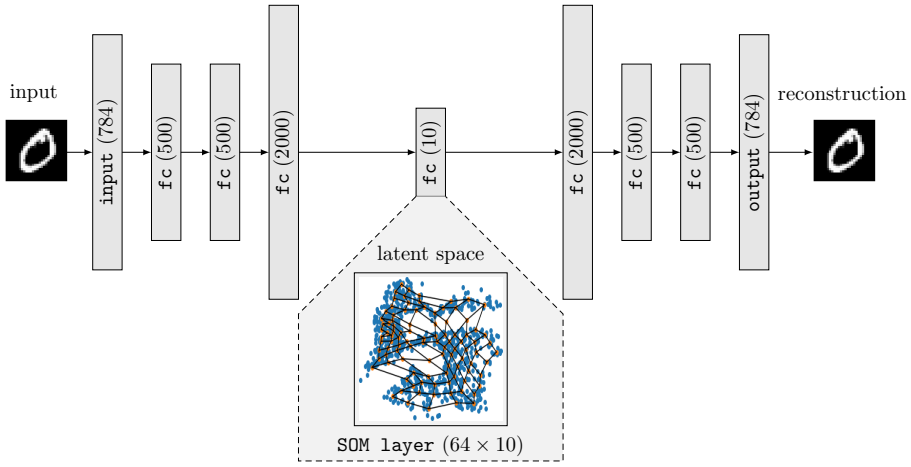

Fig. 1. Architecture of DESOM layers with an 8×8 map and 10-dimensional latent.

by stochastic gradient descent. The architecture of DESOM, in the case of a fully-connected AE dimensioned for the MNIST dataset, a 10-dimensional latent space and an 8×8 map, is illustrated in figure 1.

2.1 Loss function

We note $\mathbb{X} = \{\mathbf{x}_i\}_{1 \leq i \leq N}$ the data samples. The self-organizing map is composed of K units, associated with the set of prototype vectors $\{\mathbf{m}_k\}_{1 \leq k \leq K}$. $\delta(k, l)$ is the topographic distance between units k and l on the map (Manhattan distance for a 2D grid). We define the neighborhood function of the SOM and a temperature parameter T, controlling the radius of the neighborhood. In this work, we adopt a gaussian neighborhood: $\mathcal{K}^T(d) = e^{-\frac{d^2}{T^2}}$, and exponential temperature decay.

The encoder and decoder parameter weights are respectively noted $\mathbf{W_e}$ and $\mathbf{W_d}$. The encoding function is denoted by $\mathbf{f_{W_e}}$ and the decoding function by $\mathbf{g_{W_d}}$. Thus, $\mathbf{z}_i = \mathbf{f_{W_e}}(\mathbf{x}_i)$ is the embedded version of $\mathbf{x}_i$ in the intermediate latent space, and $\tilde{\mathbf{x}}_i = \mathbf{g_{W_d}}(\mathbf{f_{W_e}}(\mathbf{x}_i))$ is its reconstruction by the decoder. Our goal is to jointly optimize the autoencoder network weights and the SOM prototype vectors. For this task, we define a loss function composed of two terms, that can be written as:

$$\mathcal{L}(\mathbf{W_e}, \mathbf{W_d}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \chi) = \mathcal{L}_r(\mathbf{W_e}, \mathbf{W_d}) + \gamma \mathcal{L}_{som}(\mathbf{W_e}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \chi) \quad (1)$$

The first term $\mathcal{L}_r$ is the autoencoder reconstruction loss, chosen to be a simple least squares loss:

$$\mathcal{L}_r(\mathbf{W_e}, \mathbf{W_d}) = \sum_i ||\tilde{\mathbf{x}}_i - \mathbf{x}_i||^2 \quad (2)$$

The second term is the self-organizing map loss, denoted $\mathcal{L}_{som}$. It depends on the set of parameters $\{\mathbf{m}_k\}_{1 \leq k \leq K}$ and on the assignement function, denoted χ, assigning a data point to its closest prototype according to euclidean distance, i.e.:

$$\chi(\mathbf{z}) = \underset{k}{\operatorname{argmin}} ||\mathbf{z} - \mathbf{m}_k||^2 \tag{3}$$

The expression of the self-organizing map loss is:

$$\mathcal{L}_{som}(\mathbf{W_e}, \mathbf{m}_1, \ldots, \mathbf{m}_K, \chi) = \sum_i \sum_{k=1}^{K} \mathcal{K}^T \left(\delta(\chi(\mathbf{z}_i), k)\right) ||\mathbf{z}_i - \mathbf{m}_k||^2 \tag{4}$$

Note that when T converges towards zero, the SOM loss becomes identical to a k-means loss, thus our model converges towards a model equivalent to DCN [18] or DKM [3]. Finally, the hyperparameter γ trades off between minimizing the autoencoder reconstruction loss and the self-organizing map loss.

2.2 Gradients and training

We use a joint training procedure optimizes both the network parameters and the prototypes using stochastic gradient descent (with the Adam optimization scheme [9]), as the $\mathcal{L}_r$ loss is differentiable; the only non-differentiable parts are the weighting terms $w_{i,k} \equiv \mathcal{K}^T \left(\delta(\chi(\mathbf{z}_i), k)\right)$ of the SOM loss. To alleviate this, we compute the best matching units for the current (encoded) batch and fix the assignment function χ between each optimization step. Thus, these terms $w_{i,k}$ become constant with respect to the network parameters and the prototypes. This requires to compute the pairwise euclidean distances between the map prototypes and the current batch of (encoded) samples between each SGD step. The gradients of the loss function $\mathcal{L}$ w.r.t. autoencoder weights and prototypes are easy to derive if we consider the assignment function to be fixed at each step:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W_e}} = \frac{\partial \mathcal{L}_r}{\partial \mathbf{W_e}} + \gamma \frac{\partial \mathcal{L}_{som}}{\partial \mathbf{W_e}} \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W_d}} = \frac{\partial \mathcal{L}_r}{\partial \mathbf{W_d}} \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{m}_k} = \gamma \frac{\partial \mathcal{L}_{som}}{\partial \mathbf{m}_k} \tag{7}$$

The gradients for a single data point $\mathbf{x}_i$ are:

$$\frac{\partial \mathcal{L}_r^i}{\partial \mathbf{W_e}} = 2(\mathbf{g_{W_d}}(\mathbf{f_{W_e}}(\mathbf{x}_i)) - \mathbf{x}_i)\frac{\partial \mathbf{g_{W_d}}(\mathbf{f_{W_e}}(\mathbf{x}_i))}{\partial \mathbf{W_e}} \tag{8}$$

$$\frac{\partial \mathcal{L}_r^i}{\partial \mathbf{W_d}} = 2(\mathbf{g_{W_d}}(\mathbf{f_{W_e}}(\mathbf{x}_i)) - \mathbf{x}_i)\frac{\partial \mathbf{g_{W_d}}(\mathbf{f_{W_e}}(\mathbf{x}_i))}{\partial \mathbf{W_d}} \tag{9}$$

$$\frac{\partial \mathcal{L}_{som}^i}{\partial \mathbf{W_e}} = 2\sum_{k=1}^{K} w_{i,k}(\mathbf{f_{W_e}}(\mathbf{x}_i) - \mathbf{m}_k)\frac{\partial \mathbf{f_{W_e}}(\mathbf{x}_i)}{\partial \mathbf{W_e}} \tag{10}$$

$$\frac{\partial \mathcal{L}_{som}^i}{\partial \mathbf{m}_k} = 2w_{i,k}(\mathbf{m}_k - \mathbf{f_{W_e}}(\mathbf{x}_i)) \tag{11}$$

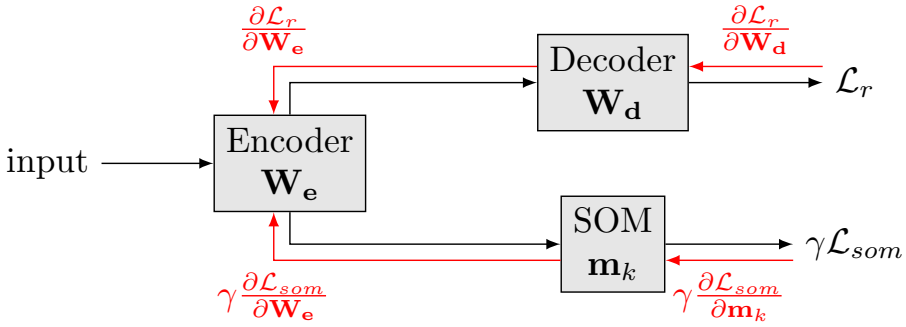

Fig. 2. Path of gradients in the DESOM model.

The paths of the gradients of the loss function are illustrated on figure 2. We optimize 1 using backpropagation and minibatch stochastic gradient descent (SGD), with a learning rate l_r (in our experiments Adam is used instead, but the equations are derived for vanilla SGD). Given a batch of n_b samples, the encoder's weights are updated by:

$$\mathbf{W_e} \leftarrow \mathbf{W_e} - \frac{l_r}{n_b}\sum_{i=1}^{n_b}\left(\frac{\partial \mathcal{L}_r^i}{\partial \mathbf{W_e}} + \gamma\frac{\partial \mathcal{L}_{som}^i}{\partial \mathbf{W_e}}\right) \tag{12}$$

The decoder's weights are updated by:

$$\mathbf{W_d} \leftarrow \mathbf{W_d} - \frac{l_r}{n_b}\sum_{i=1}^{n_b}\frac{\partial \mathcal{L}_r^i}{\partial \mathbf{W_d}} \tag{13}$$

And finally, the map prototypes are updated by the following update rule:

$$\mathbf{m}_k \leftarrow \mathbf{m}_k - \frac{l_r}{n_b}\sum_{i=1}^{n_b}\gamma\frac{\partial \mathcal{L}_{som}^i}{\partial \mathbf{m}_k} \tag{14}$$

2.3 Training procedure

The training procedure is detailed in algorithm 1.

Input: training set $\mathbb{X}$; SOM dimensions (m, n); initial and final temperatures T_{max}, T_{min}; number of iterations *iterations*; batch size *batchSize*
Output: AE weights $\mathbf{W_e}$, $\mathbf{W_d}$; SOM code vectors $\{\mathbf{m}_k\}$
Initialize AE weights $\mathbf{W_e}$, $\mathbf{W_d}$ (Glorot uniform scheme) ;
Initialize SOM parameters $\{\mathbf{m}_k\}$ (with random data sample) ;
for $iter = 1, \ldots,$ *iterations* **do**
  $T \leftarrow T_{max} \left(\frac{T_{min}}{T_{max}}\right)^{iter/iterations}$;
  Load next training batch ;
  Encode current batch and compute weights $w_{i,k}$;
  Train DESOM on batch by taking a SGD step (by 12, 13 and 14) ;
end

Algorithm 1: DESOM training procedure

3 Evaluation

We evaluated the clustering and visualization performance of our model on standard classification benchmark datasets. In this section, we will detail our evaluation methodology and the results on the MNIST and REUTERS-10k datasets.

Datasets The MNIST dataset [12] consists in 70000 grayscale images of handwritten digits, of size 28-by-28 pixels. We divided pixel intensities by 255 to obtain a 0-1 range and flattened the images into 784-dimensional vectors. REUTERS-10k [13] is a text dataset built from the RCV1-v2 corpus. REUTERS-10k is created by restricting the documents to 4 classes, sampling a subset of 10000 examples and computing TF-IDF features on the 2000 most frequently occurring words. We used the same preparation code as in [5].

Qualitative evaluation We assessed that, just like a standard SOM, our model produces a topologically organized map for efficient visualization, and that the decoded code vectors are of high quality. An example of DESOM map for MNIST can be seen on figure 3. We verified that the capacity of the AE (number of layers and units) was directly linked with the visual quality of the prototypes. In particular, using standard SOM directly on this kind of data produces blurred prototype images, due to averaging in original space.

Quantitative evaluation Then, we evaluated the clustering quality of DESOM by measuring two common external clustering indices, purity and NMI (Normalized Mutual Information). We compared DESOM with 5 other SOM-based models:

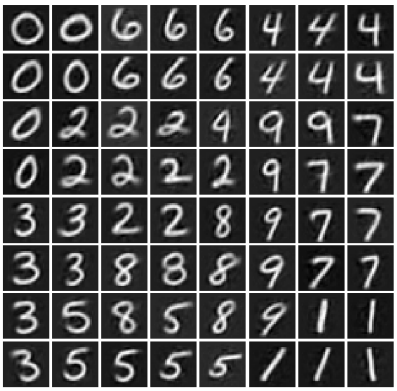

Fig. 3. Decoded prototypes visualized on a DESOM map for MNIST.

- **minisom**: a standard SOM (from minisom module[3]).
- **kerasom**: our implementation of a SOM in Keras (equivalent to DESOM with identity encoder and decoder) and trained by SGD.
- **AE+minisom**: a two-stage approach where minisom is fit on the encoded dataset using an autoencoder with the same architecture as the one used in DESOM.
- **AE+kerasom**: the same two-stage approach but with our kerasom model, resulting in DESOM without joint optimization of AE and SOM.
- **SOM-VAE**: results from the author's paper [4] (only for MNIST).
- **DESOM**: our proposed DESOM model with joint representation learning and self-organization.

In all models, the AE has a $[500, 500, 2000, 10]$ architecture and the map has 8×8 units. The γ parameter was fixed empirically to 0.001, number of iterations is 10000 with a batch size of 256. Results are summarized in table 1. Results on MNIST show that reducing dimensionality with an autoencoder improves clustering quality. DESOM and AE+kerasom perform best and have similar results, but DESOM requires no pre-training. However, the AE struggles to find good representations on REUTERS-10k, and DESOM outperforms all other models by a large margin, suggesting that joint training with a self-organizing map prior has enabled to learn SOM-friendly representations.

Training time An advantage of joint training is reduced training time of DESOM compared with AE+kerasom (other models cannot be compared due to difference in implementation). Indeed, to obtain the results listed in table 1, kerasom and DESOM were trained for the same number of iterations and required the same training time (about 2 minutes on MNIST on a laptop GPU). If

[3] `https://github.com/JustGlowing/minisom`

Table 1. Clustering performance of SOM-based models according to purity and NMI (mean and standard deviation on 10 runs). Best performance in bold.

	MNIST		REUTERS-10k	
Method	purity	NMI	purity	NMI
minisom (8×8)	0.637 ± 0.023	0.430 ± 0.016	0.690 ± 0.028	0.230 ± 0.024
kerasom (8×8)	0.826 ± 0.005	0.565 ± 0.003	0.697 ± 0.067	0.324 ± 0.051
AE+minisom (8×8)	0.872 ± 0.017	0.616 ± 0.010	0.690 ± 0.021	0.235 ± 0.015
AE+kerasom (8×8)	**0.939** ± 0.003	**0.661** ± 0.002	0.777 ± 0.012	0.306 ± 0.010
SOM-VAE (8×8)	0.868 ± 0.003	0.595 ± 0.002	-	-
DESOM (8×8)	**0.939** ± 0.004	0.657 ± 0.004	**0.849** ± 0.011	**0.381** ± 0.009

we add the AE pretraining time, the overall training time of AE+kerasom nearly doubles (we pretrained for 200 epochs). As a conclusion, the joint representation learning adds almost no training time overhead.

4 Architecture variants

In this section, we present extensions of the generic DESOM architecture for data with structural regularities such as images and sequential data (e.g. multivariate timeseries). The advantage of deep architectures is that we only need to change the representation learning part of the model (autoencoder), that maps the input to its latent embedding; the SOM layer, training procedure and loss function remain the same.

4.1 ConvDESOM: convolutions for images and sequences

For image data, we can replace the fully-connected autoencoder with a convolutional autoencoder and a typical architecture for image recognition, composed of alternating 2D convolutions and pooling operations. An example of such an architecture, that we call ConvDESOM, is represented on figure 4. The output of the encoder is now a 2D feature map that is flattened before serving as input to the SOM layer. The decoder is composed of convolutional and up-sampling layers.

For sequence data and time series in particular, the same type of architecture can be used, but with 1D convolutions instead. The exact architecture depends on the use case:

- Pooling layers will mitigate location dependance.
- Convolutions may be causal and/or dilated (dilation allows to increase the receptive field exponentially with the network depth while keeping the number of parameters low, thus allowing for long effective memory [2]).
- Convolution kernel size, dilation, padding policy, pooling and the number of layers have a direct influence on the dimensionality of the latent code used by the SOM layer.

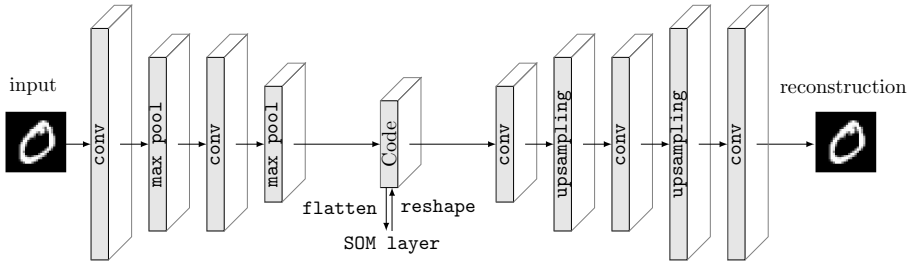

Fig. 4. Architecture of the ConvDESOM model variant (example for 2D images).

We are still conducting experiments on these architectures.

4.2 LSTM-DESOM: recurrent architecture for sequences

In this last section, we propose a recurrent variant of DESOM, called LSTM-DESOM, based on Long Short-Term Memory (LSTM) cells [7]. This architecture is targeting sequential data, and like causal convolutions, recurrent networks can handle sequences of arbitrary length. It is based on an LSTM autoencoder [16], which is a particular case of the LSTM encoder-decoder architecture that learns a code representation from an input sequence and then reconstructs this sequence. Similarly to the standard DESOM presented in the second section, the latent representation is used to learn the SOM code vectors. An unrolled illustration of a basic LSTM-DESOM architecture is represented in figure 5.

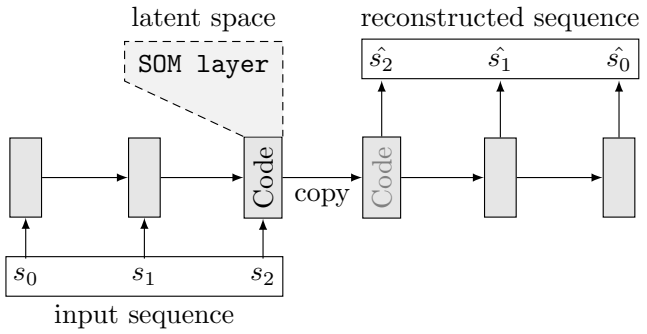

Fig. 5. Unrolled architecture of the LSTM-DESOM model variant.

In practice, the model can have multiple layers, and may condition the decoder on the reversed input sequence for reconstruction (see [16]). A slightly more complex architecture is used in [14] for joint representation learning and

clustering of time series. Again, no experiments with this architecture have been conducted yet.

5 Conclusion and future work

The Deep Embedded Self-Organizing Map extends the ideas of joint representation learning and clustering to topology-preserving clustering with self-organizing maps. It can be used to explore and visualize large, high-dimensional datasets, and the architecture can be adapted for various types of data, including images and sequences. Compared with other SOM-based algorithms, it shows similar or superior performance. By combining representation learning and self-organization in a joint task, it reduces overall training time compared with traditional two-stage approaches. The specific properties of the *SOM-friendly* latent space learned by DESOM need to be studied more thoroughly in future work. Future work will also include the study and evaluation of the convolutional and recurrent variants of DESOM.

References

1. Aljalbout, E., Golkov, V., Siddiqui, Y., Cremers, D.: Clustering with Deep Learning: Taxonomy and New Methods (2018), http://arxiv.org/abs/1801.07648
2. Bai, S., Kolter, J.Z., Koltun, V.: An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling (2018). https://doi.org/10.1016/S0925-5273(03)00047-1, http://arxiv.org/abs/1803.01271
3. Fard, M.M., Thonet, T., Gaussier, E.: Deep k-Means: Jointly Clustering with k-Means and Learning Representations (2018), http://arxiv.org/abs/1806.10069
4. Fortuin, V., Hüser, M., Locatello, F., Strathmann, H., Rätsch, G.: Deep Self-Organization: Interpretable Discrete Representation Learning on Time Series (2018), http://arxiv.org/abs/1806.02199
5. Guo, X., Gao, L., Liu, X., Yin, J.: Improved deep embedded clustering with local structure preservation. In: IJCAI. pp. 1753–1759 (2017)
6. Harchaoui, W., Mattei, P., Alamansa, A., Bouveyron, C.: Wasserstein Adversarial Mixture Clustering (2018), https://hal.archives-ouvertes.fr/hal-01827775/
7. Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Computation **9**(8), 1735–1780 (1997)
8. Jiang, Z., Zheng, Y., Tan, H., Tang, B., Zhou, H.: Variational Deep Embedding : An Unsupervised and Generative Approach to Clustering. In: IJCAI. pp. 1965–1972 (2017). https://doi.org/10.24963/ijcai.2017/273
9. Kingma, D.P., Ba, J.L.: Adam: A Method For Stochastic Optimization. In: ICLR (2015), http://arxiv.org/abs/1412.6980
10. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biological Cybernetics **43**(1), 59–69 (1982). https://doi.org/10.1007/BF00337288
11. Kohonen, T.: The Self-Organizing Map. In: Proceedings of the IEEE. vol. 78, pp. 1464–1480 (1990). https://doi.org/10.1109/5.58325
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based Learning Applied to Document Recognition. In: Proceedings of the IEEE (1998). https://doi.org/10.1109/5.726791

13. Lewis, D.D., Yang, Y., Rose, T.G., Li, F.: RCV1: A New Benchmark Collection for Text Categorization Research. Journal of Machine Learning Research **5**, 361–397 (2004), http://dl.acm.org/citation.cfm?id=1005332.1005345
14. Madiraju, N.S., Sadat, S.M., Fisher, D., Karimabadi, H.: Deep Temporal Clustering : Fully Unsupervised Learning of Time-Domain Features pp. 1–11 (2018), http://arxiv.org/abs/1802.01059
15. van den Oord, A., Vinyals, O., Kavukcuoglu, K.: Neural Discrete Representation Learning. In: NIPS (2017), http://arxiv.org/abs/1711.00937
16. Srivastava, N., Mansimov, E., Salakhutdinov, R.: Unsupervised Learning of Video Representations using LSTMs (2015). https://doi.org/citeulike-article-id:13519737, http://arxiv.org/abs/1502.04681
17. Xie, J., Girshick, R., Farhadi, A.: Unsupervised Deep Embedding for Clustering Analysis. In: ICML. vol. 48 (2015). https://doi.org/10.1007/JHEP01(2016)157, http://arxiv.org/abs/1511.06335
18. Yang, B., Fu, X., Sidiropoulos, N.D., Hong, M.: Towards K-means-friendly Spaces: Simultaneous Deep Learning and Clustering. In: ICML (2016), http://arxiv.org/abs/1610.04794
19. Yang, J., Parikh, D., Batra, D.: Joint Unsupervised Learning of Deep Representations and Image Clusters (2016). https://doi.org/10.1109/CVPR.2016.556, http://arxiv.org/abs/1604.03628